

Лабораторная работа №9. JavaDB

Теоретическая часть

Средства для работы с БД в Java

Основная библиотека для работы с базами данных в Java это `java.sql.*`. Далее рассмотрим её основные классы, интерфейсы и методы.

Класс *Connection*

Класс `java.sql.Connection` представляет в JDBC сеанс работы с базой данных. Он предоставляет приложению объекты `Statement` (и его подклассы) для этого сеанса. Он также управляет транзакциями для этих команд. По умолчанию каждая команда выполняется в отдельной транзакции. Объект `Connection` позволяет отключить функцию `Autocommit` автоматического завершения транзакции. В этом случае требуется явно завершить транзакцию, иначе результаты выполнения всех команд будут потеряны.

Основные методы класса *Connection*

`void close() throws SQLException`

Этот метод позволяет вручную освободить все ресурсы, такие как сетевые соединения и блокировки базы данных, связанные с данным объектом `Connection`. Этот метод автоматически вызывается при сборке мусора; лучше, однако, вручную закрыть `Connection`, если вы в нем больше не нуждаетесь.

`Statement createStatement() throws SQLException`

`Statement createStatement(int resultSetType, int resultSetConcurrency) throws SQLException`
`Statement createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException`

Метод создаёт объект `Statement`, связанный с сеансом `Connection`. Версия без аргументов создаёт объект `Statement`, для которого экземпляры `ResultSet` имеют тип только для чтения и перемещения в прямом направлении.

`boolean getAutoCommit() throws SQLException`
`void setAutoCommit(boolean ac) throws SQLException`

По умолчанию все объекты `Connection` находятся в режиме автозавершения. В этом режиме каждая команда завершается сразу после выполнения. Может оказаться предпочтительнее вручную завершить серию команд в приложении как единую транзакцию. В этом случае метод `setAutoCommit()` используется для отключения автозавершения. Затем, после выполнения своих команд, вы вызываете `commit()` или `rollback()`, в зависимости от успеха или неуспеха транзакции. В режиме автозавершения команда завершается, когда она выполнена, или выполняется следующая команда, в зависимости от того, что произойдёт раньше. Команда, возвращающая `ResultSet`, выполнена, когда извлечена последняя строка или закрыт объект `ResultSet`. Если команда возвращает множественные результирующие наборы, завершение происходит после извлечения последней строки последнего объекта `ResultSet`.

void commit() throws SQLException

Этот метод делает постоянными изменения, произведённые всеми командами, связанными с данным соединением и выполненными вслед за последней командой завершения или отката транзакции. Использовать его следует только при отключённом автозавершении. Он не завершает изменения, сделанные командами, которые связаны с другими объектами Connection.

String getCatalog() throws SQLException

void setCatalog(String catalog) throws SQLException

Если драйвер поддерживает каталоги, то setCatalog() используется для выбора подпространства базы данных с заданным именем каталога. Если драйвер каталоги не поддерживает, запрос игнорируется.

DatabaseMetaData getMetaData() throws SQLException

Класс DatabaseMetaData предоставляет методы, описывающие таблицы базы данных, поддержку SQL, хранимые процедуры и другие сведения, относящиеся к базе данных и данному Connection, которые не относятся непосредственно к выполнению команд и извлечению результирующих наборов. Метод создаёт экземпляр класса DatabaseMetaData для данного Connection.

SQLWarning getWarnings() throws SQLException

Возвращает первое предупреждение из списка, связанного с данным объектом Connection.

Класс Statement

Класс представляет встроенную команду SQL и используется приложением для доступа к базе данных. При закрытии Statement автоматически закрываются все связанные с ним открытые объекты ResultSet.

Основные методы класса Statement

void addBatch(String sql) throws SQLException

Добавляет заданную команду SQL к текущему пакету команд.

void cancel() throws SQLException

В многопоточной среде с помощью этого метода можно потребовать прекращения всякой обработки, связанной с данным Statement. В этом смысле метод аналогичен методу stop() для объектов Thread.

boolean execute(String sql) throws SQLException

ResultSet executeQuery(String sql) throws SQLException

int executeUpdate(String sql) throws SQLException

Выполняет Statement, передавая базе данных заданную SQL-строку. Первый метод, execute(), позволяет вам выполнить Statement, когда неизвестно заранее, является SQL-строка запросом или обновлением. Метод возвращает true, если команда создала результирующий набор. Метод executeQuery() используется для выполнения запросов (на извлечение данных). Он возвращает для обработки результирующий набор. Метод executeUpdate() используется для выполнения обновлений. Он возвращает количество обновлённых строк.

`int[] executeBatch(String sql) throws SQLException`

Посылает базе данных пакет SQL-команд для выполнения. Возвращает массив чисел, описывающих количество строк, затронутых каждой командой SQL.

`ResultSet getResultSet() throws SQLException`

Метод возвращает текущий `ResultSet`. Для каждого результата его следует вызывать только однажды. Его не нужно вызывать после обращения к `executeQuery()`, возвращающему единственный результат.

`void close() throws SQLException`

Вручную закрывает объект `Statement`. Обычно этого не требуется, так как `Statement` автоматически закрывается при закрытии связанного с ним объекта `Connection`. К сожалению, не все разработчики JDBC-драйверов придерживаются этих конвенций.

Класс `ResultSet`

Этот класс представляет результирующий набор базы данных. Он обеспечивает приложению строчный доступ к результатам запросов в базе данных. Во время обработки запроса `ResultSet` поддерживает указатель на текущую обрабатываемую строку. Приложение последовательно перемещается по результатам, пока они не будут все обработаны или не будет закрыт `ResultSet`.

Основные методы класса `ResultSet`

`boolean absolute(int row) throws SQLException`

Метод перемещает курсор на заданное число строк от начала, если число положительно, и от конца - если отрицательно.

`void afterLast() throws SQLException`

Этот метод перемещает курсор в конец результирующего набора за последнюю строку.

`void beforeFirst() throws SQLException`

Этот метод перемещает курсор в начало результирующего набора перед первой строкой.

`void deleteRow() throws SQLException`

Удаляет текущую строку из результирующего набора и базы данных.

`ResultSetMetaData getMetaData() throws SQLException`

Предоставляет объект метаданных для данного `ResultSet`. Класс `ResultSetMetaData` содержит информацию о результирующей таблице, такую как количество столбцов, их заголовки и т.д.

`int getRow() throws SQLException`

Возвращает номер текущей строки.

`Statement getStatement() throws SQLException`

Возвращает экземпляр Statement, который произвёл данный результирующий набор.

```
boolean next() throws SQLException  
boolean previous() throws SQLException
```

Эти методы позволяют переместиться в результирующем наборе на одну строку вперёд или назад. Во вновь созданном результирующем наборе курсор устанавливается перед первой строкой, поэтому первое обращение к методу next() влечёт позиционирование на первую строку. Эти методы возвращают true, если остаётся строка для дальнейшего перемещения. Если строк для обработки больше нет, возвращается false. Если открыт поток InputStream для предыдущей строки, он закрывается. Также очищается цепочка предупреждений SQLWarning.

```
void close() throws SQLException
```

Осуществляет немедленное закрытие ResultSet вручную. Обычно этого не требуется, так как закрытие Statement, связанного с ResultSet, автоматически закрывает ResultSet. К сожалению, не все разработчики JDBC-драйверов придерживаются этих конвенций, например, драйвер Oracle самостоятельно не закрывает ResultSet'ы, так что настоятельно рекомендуется закрывать вручную.

PreparedStatement

Интерфейс PreparedStatement наследует от Statement и отличается от последнего следующим:

1. Экземпляры PreparedStatement "помнят" скомпилированные SQL-выражения. Именно поэтому они называются "prepared" ("подготовленные").
2. SQL-выражения в PreparedStatement могут иметь один или более входной (IN) параметр. Входной параметр - это параметр, чьё значение не указывается при создании SQL-выражения. Вместо него в выражении на месте каждого входного параметра ставится знак ("?"). Значение каждого вопросительного знака устанавливается методами setXXX перед выполнением запроса.

Поскольку объекты PreparedStatement прекомпилированы, исполнение этих запросов может происходить несколько быстрее, чем в объектах Statement. В результате SQL-выражения, которые исполняются часто, в целях улучшения производительности создают в виде объектов PreparedStatement.

Оставаясь подклассом класса Statement, класс PreparedStatement наследует все функции от Statement. В дополнении к ним он добавляет методы установки входных параметров. Кроме того, три метода - execute, executeQuery и executeUpdate - модифицированы таким образом, что не имеют аргументов. Старые методы класса Statement (которые принимают SQL-выражения в качестве единственного аргумента) не должны использоваться в объекте PreparedStatement.

Создание объектов

Следующий фрагмент кода, где con - это объект Connection, создаёт объект PreparedStatement, содержащий SQL-выражение с двумя параметрами:

```
PreparedStatement pstmt =  
    con.prepareStatement("UPDATE table4 SET m = ? WHERE x = ?");
```

Объект pstmt отныне содержит выражение "UPDATE table4 SET m = ? WHERE x = ?", которое уже отослано в СУБД и подготовлено для выполнения.

Передача входных (IN) параметров

Перед выполнением объекта PreparedStatement надо установить значения всех его параметров. Это делается с помощью методов setXXX, где XXX - это тип параметра.

Например, если параметр имеет Java-тип `long`, используемый метод будет `setLong`. Первый аргумент методов `setXXX` - это порядковый номер параметра, а второй - значение, в которое надо его установить. Например, следующий код устанавливает первый параметр в значение 123456789, а второй - в 1000000000:

```
pstmt.setLong(1, 123456789);
pstmt.setLong(2, 1000000000);
```

После установки параметра его можно использовать при многократном выполнении выражения до тех пор, пока он не очистится методом `clearParameters`.

В режиме соединения по умолчанию (разрешена автофиксация) каждый запрос фиксируется или откатывается автоматически.

Один и тот же объект `PreparedStatement` может выполняться много раз, если нижестоящий драйвер или СУБД будут сохранять выражение (`statement`) в открытом состоянии даже после того как произойдёт фиксация. Иначе не имеет смысла пытаться улучшить производительность заменой `Statement` на `PreparedStatement`.

Используя `pstmt` из предыдущего примера, следующий код устанавливает значения обоих параметров и выполняет `pstmt` 10 раз. Как уже было отмечено, БД не должна закрывать `pstmt`. В этом примере первый параметр устанавливается в "Hi" и остаётся неизменным. Второй параметр устанавливается в последовательные целые значения, начиная от 0 и заканчивая 9.

```
pstmt.setString(1, "Hi");
for (int i = 0; i < 10; i++)
{
    pstmt.setInt(2, i);
    int rowCount = pstmt.executeUpdate();
}
```

Совместимость типов данных входных параметров

`XXX` в названии методов `setXXX` - это тип данных Java. Неявно он же является и типом данных JDBC (SQL), так как драйвер отображает тип данных Java на соответствующий JDBC-тип перед отсылкой JDBC-типа в БД. Следующий код устанавливает второй параметр объекта `PreparedStatement pstmt` в 44 с типом данных `short`:

```
pstmt.setShort(2, 44);
```

Драйвер отошлёт 44 в БД в виде типа JDBC `SMALLINT`, который является стандартным для Java-типа `short`.

На программисте лежит ответственность за совместимость Java-типов входных параметров и ожидаемых базой данных соответствующих JDBC-типов. Рассмотрим случай, когда ожидается `SMALLINT`. Если используется метод `setByte`, то драйвер отошлёт значение `TINYINT` в БД. Это, вероятно, работать будет, так как многие БД преобразуют "похожие" типы данных друг в друга. Тем не менее, чтобы приложение могло работать как можно с большим количеством СУБД, лучше использовать Java-типы, которые в точности соответствуют ожидаемым JDBC-типам. Если ожидается JDBC-тип данных `SMALLINT`, то использование именно `setShort` вместо `setByte` сделает приложение более кроссплатформенным.

Использование объектов

Программист может явно задать конвертирование входных параметров в определённый JDBC-тип с помощью метода `setObject`. Этот метод может принимать третий аргумент, указывающий целевой JDBC-тип данных. Перед отправкой в БД драйвер преобразует `Object` в указанный JDBC-тип.

Если JDBC-тип не задан, то драйвер просто преобразует Object в его ближайший JDBC-эквивалент, а затем пошлёт его в БД. Это подобно использованию обычных методов setXXX; в обоих случаях драйвер отображает Java-типы в JDBC-типы данных перед отправкой значений в БД. Разница заключается в том, что если методы setXXX используют таблицу отображения Java-типов в JDBC-типы, то метод setObject использует несколько отличную таблицу отображения.¹

В случае использования метода setObject тип данных может быть неизвестен приложению на этапе его компиляции. Используя его, приложение может подавать на вход значения любых типов данных и преобразовывать их в ожидаемый базой данных JDBC-тип.

Отсылка значений NULL в качестве входного параметра

Метод setNull позволяет отсылать значения NULL в БД как входные параметры. Хотя JDBC-тип параметра все же можно задать явно. JDBC-значение NULL будет отослано в БД также в том случае, если методу setXXX будет передано Java-значение null (если метод принимает Java-объект в качестве аргумента). Тем не менее, метод setObject может принять значение null только в случае, если задан JDBC-тип.

Отправка очень больших входных параметров

Методы setBytes и setString могут отсылать неограниченное количество данных. Правда, иногда программисту легче передавать большие значения в виде маленьких кусков. Это делается установкой входного параметра в значение Java-потока ввода (input stream). Когда выполняется выражение, JDBC-драйвер будет производить последовательные вызовы из этого потока ввода, считывая его содержимое и пересылая его в виде значения параметра. JDBC предоставляет три метода установки входных параметров в поток ввода:

- setBinaryStream для потоков, содержащих обычные байты.
- setAsciiStream для потоков ASCII-символов.
- setUnicodeStream для потоков Unicode-символов.

Эти методы, в отличие от остальных методов setXXX, принимают дополнительный аргумент, равный количеству передаваемых в потоке байтов. Этот аргумент необходим, так как некоторые СУБД требуют указания размера данных перед их отсылкой.

Следующий код иллюстрирует отсылку файла в виде входного параметра запроса:

```
java.io.File file = new java.io.File("/tmp/data");
int fileLength = file.length();

java.io.InputStream fin = new java.io.FileInputStream(file);

java.sql.PreparedStatement pstmt = con.prepareStatement(
    "UPDATE Table5 SET stuff = ? WHERE index = 4");

pstmt.setBinaryStream (1, fin, fileLength);
pstmt.executeUpdate();
```

При выполнении запроса для доставки данных последовательно считывается поток ввода fin.

Транзакций при работе с БД в Java

Отключение режима автоматической фиксации

Когда соединение с базой данных создано, оно находится в режиме автоматической фиксации. Это означает, что каждый отдельный оператор SQL рассматривается как транзакция и автоматически фиксируется сразу после её выполнения. Чтобы разрешить группировку нескольких операторов в транзакцию, необходимо отключить режим

1 https://download.oracle.com/otn_hosted_doc/jdeveloper/904preview/jdk14doc/docs/guide/jdbc/getstart/mapping.html

автоматической фиксации. Это демонстрируется в следующем коде, где con - активное соединение:

```
Con.setAutoCommit (false);
```

Выполнение транзакций

После того, как режим автоматической фиксации отключён, никакие инструкции SQL не выполняются до тех пор, пока вы не вызовете метод commit явно. Все операторы, выполненные после предыдущего вызова метода commit, включаются в текущую транзакцию и передаются вместе как единое целое. Следующий метод CoffeesTable.updateCoffeeSales, в котором con является активным соединением, иллюстрирует транзакцию:

```
public void updateCoffeeSales(HashMap<String, Integer> salesForWeek)
    throws SQLException {

    PreparedStatement updateSales = null;
    PreparedStatement updateTotal = null;

    String updateString = "update " + dbName + ".COFFEES " +
        "set SALES = ? where COF_NAME = ?";

    String updateStatement = "update " + dbName + ".COFFEES " +
        "set TOTAL = TOTAL + ? where COF_NAME = ?";

    try {
        con.setAutoCommit(false);
        updateSales = con.prepareStatement(updateString);
        updateTotal = con.prepareStatement(updateStatement);

        for (Map.Entry<String, Integer> e : salesForWeek.entrySet()) {
            updateSales.setInt(1, e.getValue().intValue());
            updateSales.setString(2, e.getKey());
            updateSales.executeUpdate();
            updateTotal.setInt(1, e.getValue().intValue());
            updateTotal.setString(2, e.getKey());
            updateTotal.executeUpdate();
            con.commit();
        }
    } catch (SQLException e) {
        JDBCUtilities.printSQLException(e);
        if (con != null) {
            try {
                System.err.print("Transaction is being rolled back");
                con.rollback();
            } catch (SQLException excep) {
                JDBCUtilities.printSQLException(excep);
            }
        }
    } finally {
        if (updateSales != null) {
            updateSales.close();
        }
        if (updateTotal != null) {
            updateTotal.close();
        }
        con.setAutoCommit(true);
    }
}
```


В этом методе режим автоматической фиксации отключён для подключения con, что означает, что два подготовленных оператора updateSales и updateTotal совместно передаются, когда вызывается метод commit. Всякий раз, когда вызывается метод commit (либо автоматически, когда режим авто-фиксации включён, либо явно, когда он отключён) все изменения, вызванные операциями в транзакции, становятся постоянными. В этом случае это означает, что столбцы SALES и TOTAL для колумбийского кофе были изменены.

Утверждение

```
con.setAutoCommit (true);
```

Включает режим автоматической фиксации, что означает, что каждый оператор снова автоматически передаётся после его завершения. Вы возвращаетесь в состояние по умолчанию, где вам не нужно вызывать метод commit. Целесообразно отключить режим автоматической фиксации только в режиме транзакции. Таким образом, вы избегаете хранения блокировок базы данных для нескольких операторов, что увеличивает вероятность конфликтов с другими пользователями.

Использование транзакций для сохранения целостности данных

Транзакции могут помочь сохранить целостность данных в таблице. Они обеспечивают определённый уровень защиты от конфликтов, возникающих при одновременном доступе двух пользователей к данным.

Чтобы избежать конфликтов во время транзакции, СУБД использует блокировки, механизмы для блокирования доступа других к данным, к которым обращается транзакция. (Обратите внимание, что в режиме автоматической фиксации, когда каждый оператор является транзакцией, блокировки сохраняются только для одного оператора.) После того, как блокировка установлена, она остаётся в силе до тех пор, пока транзакция не будет зафиксирована или не отменена. Например, СУБД может заблокировать строку таблицы до тех пор, пока не будут зафиксированы обновления для неё. Эффект этой блокировки будет состоять в том, чтобы предотвратить получение "грязного" чтения пользователем, то есть чтение значения до того, как оно станет постоянным.

Как устанавливаются блокировки, определяется тем, что называется уровнем изоляции транзакций, который может варьироваться от "полного отсутствия поддержки транзакций" до поддержки транзакций, которые обеспечивают соблюдение очень строгих правил доступа.

Одним из примеров уровня изоляции транзакций является

TRANSACTION_READ_COMMITTED, который не позволит получить доступ к значению до тех пор, пока оно не будет зафиксировано. Другими словами, если уровень изоляции транзакции установлен в TRANSACTION_READ_COMMITTED, СУБД не позволяет производить грязные чтения. Интерфейс Connection содержит пять значений, которые представляют уровни изоляции транзакций, которые вы можете использовать в JDBC, таблица 9.1:

Табл. 9.1 Уровни изоляции транзакций, предоставляемые JDBC

Уровень изоляции	Транзакции	Грязное чтение	Неповторяющееся чтение	Фантомное чтение
TRANSACTION_NONE	нет	нет	нет	нет
TRANSACTION_READ_COMMITTED	поддерживает	нет	да	да
TRANSACTION_READ_UNCOMMITTED	поддерживает	да	да	да
TRANSACTION_REPEATABLE_READ	поддерживает	нет	нет	да
TRANSACTION_SERIALIZABLE	поддерживает	нет	нет	нет

Грязное чтение

Происходит, когда имеет место доступ к обновлённому значению, которое не было зафиксировано, так как возможно, что это значение будет возвращено к его предыдущему значению. Если вы прочитаете значение, которое позже будет отменено, вы увидите недопустимое значение.

Неповторяющееся чтение(отсутствие повторного чтения)

Происходит, когда транзакция А извлекает строку, транзакция В впоследствии обновляет строку, а транзакция А позже извлекает эту же строку снова. Транзакция А дважды извлекает одну и ту же строку, но видит разные данные.

Фантомное чтение

Происходит, когда транзакция А извлекает набор строк, удовлетворяющих заданному условию, транзакция В впоследствии вставляет или обновляет строку, так что строка теперь соответствует условию в транзакции А, а транзакция А позже повторяет условное извлечение. В транзакции А теперь отображается дополнительная строка. Эта строка называется фантомом.

Установка уровня изоляции транзакций

Обычно вам не нужно ничего делать с уровнем изоляции транзакций. Вы можете просто использовать значение по умолчанию для вашей СУБД. Уровень изоляции транзакции по умолчанию зависит от вашей СУБД. Например, для Java DB это TRANSACTION_READ_COMMITTED. JDBC позволяет узнать, на каком уровне изоляции транзакции установлена ваша СУБД (с помощью метода Connection getTransactionIsolation), а также позволяет установить его на другой уровень (с помощью метода Connection setTransactionIsolation).

Драйвер JDBC может не поддерживать все уровни изоляции транзакций. Если драйвер не поддерживает уровень изоляции, указанный в вызове setTransactionIsolation, драйвер может заменить более высокий и более ограничительный уровень изоляции транзакций. Если драйвер не может заменить более высокий уровень транзакции, он генерирует исключение SQLException. Используйте метод DatabaseMetaData.supportsTransactionIsolationLevel, чтобы определить, поддерживает ли драйвер заданный уровень.

Настройка и возврат к точкам сохранения

Метод Connection.setSavepoint, устанавливает объект Savepoint в текущую транзакцию.

Метод Connection.rollback перегружен, чтобы использовать аргумент Savepoint.

Метод CoffeesTable.modifyPricesByPercentage повышает цену определённого сорта кофе на указанный процент, priceModifier. Однако, если новая цена больше заданной максимальной цены, то цена возвращается к первоначальной цене:

```
public void modifyPricesByPercentage(String coffeeName, float priceModifier,
                                     float maximumPrice) throws SQLException {

    con.setAutoCommit(false);

    Statement getPrice = null;
    Statement updatePrice = null;
    ResultSet rs = null;
    String query = "SELECT COF_NAME, PRICE FROM COFFEES " +
                  "WHERE COF_NAME = '" + coffeeName + "'";

    try {
        Savepoint save1 = con.setSavepoint();
        getPrice = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                       ResultSet.CONCUR_READ_ONLY);
        updatePrice = con.createStatement();
```

```

    if (!getPrice.execute(query)) {
        System.out.println("Could not find entry for coffee named " +
                           coffeeName);
    } else {
        rs = getPrice.getResultSet();
        rs.first();
        float oldPrice = rs.getFloat("PRICE");
        float newPrice = oldPrice + (oldPrice * priceModifier);
        System.out.println("Old price of " + coffeeName +
                           " is " + oldPrice);

        System.out.println("New price of " + coffeeName +
                           " is " + newPrice);

        System.out.println("Performing update...");

        updatePrice.executeUpdate("UPDATE COFFEES SET PRICE = " +
                                   newPrice + " WHERE COF_NAME = '" + coffeeName + "'");

        System.out.println("\nCOFFEES table after update:");

        CoffeesTable.viewTable(con);

        if (newPrice > maximumPrice) {
            System.out.println("\nThe new price, " + newPrice +
                               ", is greater than the maximum price, " +
                               maximumPrice + ". Rolling back the " +
                               "transaction...");

            con.rollback(save1);

            System.out.println("\nCOFFEES table after rollback:");

            CoffeesTable.viewTable(con);
        }
        con.commit();
    }
} catch (SQLException e) {
    JDBCUtilities.printSQLException(e);
} finally {
    if (getPrice != null) { getPrice.close(); }
    if (updatePrice != null) { updatePrice.close(); }
    con.setAutoCommit(true);
}
}

```

Следующая инструкция указывает, что курсор объекта `ResultSet`, сгенерированного из запроса `getPrice`, закрывается при вызове метода `commit`. Обратите внимание: если ваши СУБД не поддерживают `ResultSet.CLOSE_CURSORS_AT_COMMIT`, эта константа игнорируется:

```
GetPrice = con.prepareStatement (query, ResultSet.CLOSE_CURSORS_AT_COMMIT);
```

Метод `CoffeesTable.modifyPricesByPercentage` начинается с создания `Savepoint` с помощью следующего оператора:

```
Savepoint save1 = con.setSavepoint();
```

Метод `CoffeesTable.modifyPricesByPercentage` проверяет, превышает ли новая цена максимальное значение. Если это так, то происходит откат транзакции:

```
Con.rollback(save1);
```

Следовательно, когда метод `CoffeesTable.modifyPricesByPercentage` совершает транзакцию, вызывая метод `Connection.commit`, он не будет фиксировать никаких строк, для которых соответствующая им точка сохранения была отброшена; он будет фиксировать все остальные обновлённые строки.

Освобождение точек сохранения

Метод `Connection.releaseSavepoint` принимает объект `Savepoint` в качестве параметра и удаляет его из текущей транзакции.

После того, как точка сохранения была освобождена, попытка сослаться на неё в операции отката вызывает `SQLException`, это же касается возможных точек сохранения располагающихся после неё. Любые точки сохранения, которые были созданы в транзакции, автоматически освобождаются и становятся недействительными, когда транзакция совершается, или когда осуществляется откат всей транзакции. Возврат транзакции обратно к точке сохранения автоматически освобождает и делает недействительными любые другие точки сохранения, которые были созданы после соответствующей точки сохранения.

Когда переходить на метод обратного вызова (откат, rollback)?

Как упоминалось ранее, вызов метода отката завершает транзакцию и возвращает любые значения, которые были изменены до их предыдущих значений. Если вы пытаетесь выполнить одно или несколько операторов в транзакции и получаете `SQLException`, вызовите метод отката, чтобы завершить транзакцию и выполнить её по шагам. Это единственный способ узнать, что было выполнено и что нет. Появление `SQLException` говорит вам, что что-то не так, но оно не говорит вам, что было или не было совершено. Поскольку вы не можете рассчитывать на то, что ничего неудовлетворительного для вас не было совершено, вызов метода отката - единственный способ быть уверенным.

Метод `CoffeesTable.updateCoffeeSales` демонстрирует транзакцию и включает в себя блок `catch`, который вызывает откат метода. Если приложение продолжает использовать результаты транзакции, этот вызов метода отката в блоке `catch` предотвращает использование возможных неверных данных.

Учебное приложение

Настройка среды

1) Если у вас несколько версий JDK, то следует задать актуальное значение для переменной окружения `JAVA_HOME`.²

```
vi .bashrc
```

Добавить в его конец:

```
export JAVA_HOME=~/.soft/jdk-17.0.10/  
export PATH=$JAVA_HOME/bin:$PATH
```

2) Установить Apache Derby 10.**.*.* Release³ в зависимости от используемой версии JDK. Скачать архив `db-derby-10.**.*.*-bin.zip`.⁴

3) Запуск сервера Derby:

² <https://www.baeldung.com/java-home-on-windows-mac-os-x-linux>

Или, например, https://docs.oracle.com/cd/E19182-01/820-7851/inst_cli_jdk_javahome_t/

³ https://db.apache.org/derby/derby_downloads.html

⁴ https://db.apache.org/derby/releases/release-10_16_1_1.cgi

```
:~/soft/db-derby-10.16.1.1-bin$ ./bin/startNetworkServer
```

4) Подключить драйвер JDBC в проекте (derby.jar, derbyclient.jar, derbyshared.jar, derbytools.jar)⁵ они располагаются в архиве db-derby-10.**.*-bin.zip в папке lib, рисунки 9.1 и 9.2.

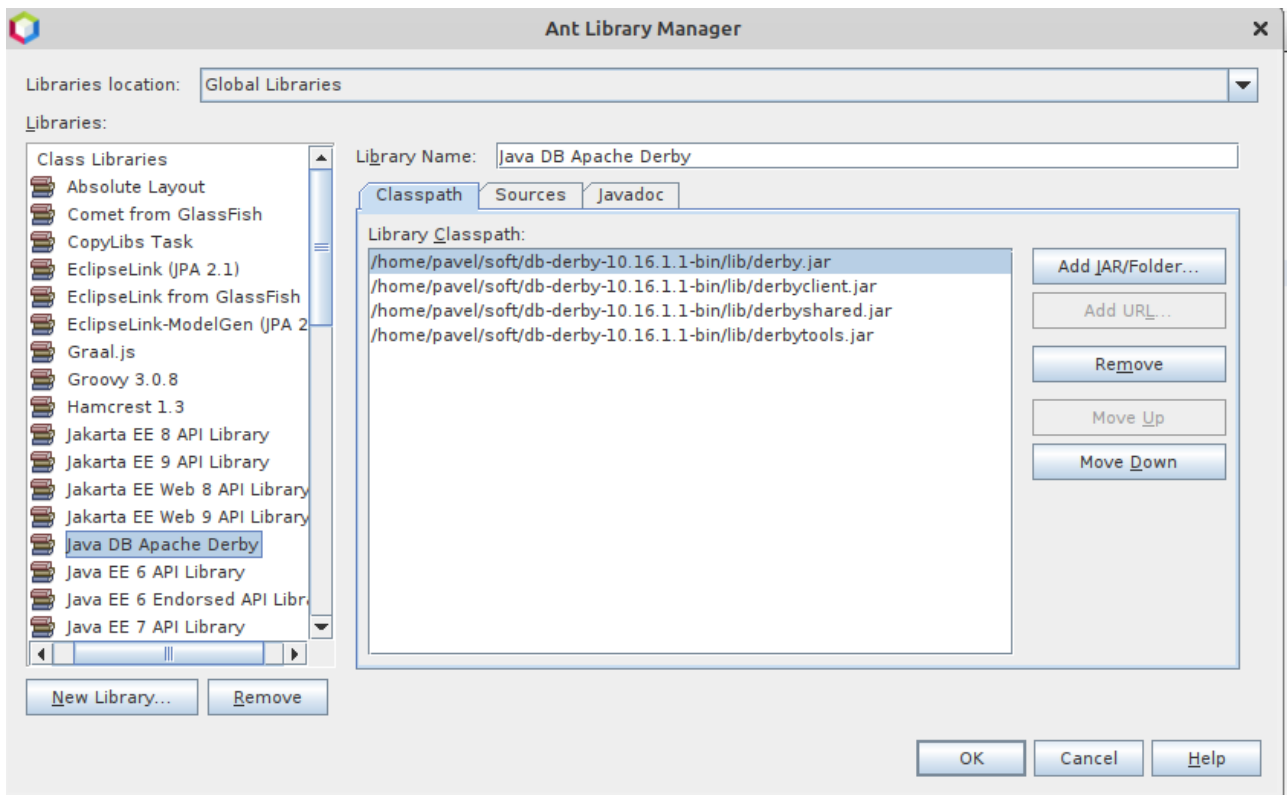


Рис. 9.1 Подключить библиотеки derby.jar, derbyclient.jar, derbyshared.jar, derbytools.jar

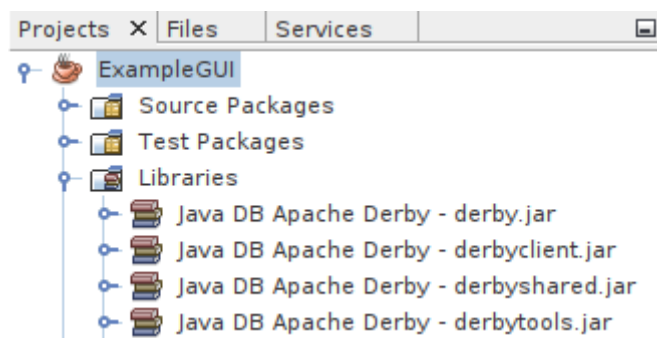


Рис. 9.2 Подключить библиотеки derby.jar, derbyclient.jar, derbyshared.jar, derbytools.jar

5) Запустить командный интерпретатор derby, чтобы создать базу данных.

```
pavel@pavel-Lenovo-M30-70:~/soft/db-derby-10.16.1.1-bin$ ./bin/ij
```

Создание базы (одной строкой):

```
CONNECT
```

```
'jdbc:derby://localhost:1527/databases/AddressBook;user=db_user;password=db_user;create=true';
```

Открытие базы (одной строкой):

```
CONNECT
```

```
'jdbc:derby://localhost:1527/databases/AddressBook;user=db_user;password=db_user';
```

⁵ На примере NetBeans.

Создание таблиц (одной строкой):

```
CREATE TABLE CONTACTS (ID_CONTACTS INT PRIMARY KEY, FIRSTNAME VARCHAR(30) NOT NULL, LASTNAME VARCHAR(30) NOT NULL, MIDDLENAME VARCHAR(30) NOT NULL, TELEPHONE INT UNIQUE NOT NULL);
```

```
CREATE TABLE PHONENUMBERS (ID_PHONENUMBER INT PRIMARY KEY, PHONENUMBER VARCHAR(12) UNIQUE NOT NULL);
```

Описание функциональности примера

В качестве примера далее приводится приложение — «Простая адресная книга». В этом приложении используется две основных панели для обеспечения отображения данных и выборки по фильтру, а также для создания новых записей, модификации и удаления существующих записей. Далее приводятся экранные формы учебного приложения, рисунок 9.3 и 9.4.

ФИО	Телефон
-----	---------

Рис. 9.3 экранные формы учебного приложения (функционал)

ФИО	Телефон
Иванов Иван Иванович	823232323
Петров Пётр Петрович	123456
Петросян Петр Петрович	800900990

Рис. 9.4 Экранные формы учебного приложения (новый контакт)

Описание базы данных учебного приложения

Описание полей таблицы "Контакты"

Table "CONTACTS":

Name: ID_CONTACTS
Date_Type: INTEGER
Size: 10
Part of the **private** key
Part of index (уникальное значение)

Name: FIRSTNAME
Date_Type: VARCHAR
Size: 30

Name: LASTNAME
Date_Type: VARCHAR
Size: 30

Name: MIDDLENAME
Date_Type: VARCHAR
Size: 30

Name: TELEPHONE
Date_Type: INTEGER
Size: 30
Part of index (уникальное значение)

Описание полей таблицы "Телефонные номера"

Table "PHONENUMBERS":

Name: ID_PHONENUMBER
Date_Type: INTEGER
Size: 10
Part of the **private** key
Part of index

Name: PHONENUMBER
Date_Type: VARCHAR
Size: 12
Part of index (уникальное значение)

Исходный код учебного приложения

```
package exampleGUI;

import java.awt.Dialog;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;
```

```

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;
import javax.swing.table.DefaultTableModel;

public class AddressBook extends JPanel implements ActionListener {
    private final int width_window = 400;
    private final int delta_size_dialog = 20;

    private static JFrame mainFrame = null;

    private static String dbURL =
        "jdbc:derby://localhost:1527/databases/AddressBook";
    private static String user_name = "db_user";
    private static String user_password = "db_user";
    private static Connection conn = null;
    private static ResultSet rs = null;
    private static Statement stmt = null;
    private static String SQL = null;

    private JPanel panelControl, panelFind, panelShow;
    private JButton buttonShow;
    private JButton buttonCreate;
    private JButton buttonEdit;
    private JButton buttonDelete;
    private JButton buttonFind;
    private JTextField textFieldFind;
    private DefaultTableModel tableShowModel;
    private JTable tableShow;
    private Object[][] data;
    private JLabel labelFindCol;

    public AddressBook() {
        super();

        try{
            DriverManager.registerDriver(new org.apache.derby.jdbc.EmbeddedDriver());
        }catch(SQLException e){

        }

        setLayout(new BoxLayout(this, BoxLayout.PAGE_AXIS));

        //Создание панели "Управление".
        panelControl = new JPanel();
        panelControl.setPreferredSize(new Dimension(width_window, 60));
        panelControl.setBorder(BorderFactory.createTitledBorder("\Управление\"));
        add(Box.createRigidArea(new Dimension(0, 10))); // Отступ 10 пикселей
        panelControl.setLayout(new FlowLayout());
        buttonShow = new JButton("Просмотреть");
        buttonShow.addActionListener(this);
        buttonCreate = new JButton("Создать");
        buttonCreate.addActionListener(this);
        buttonEdit = new JButton("Редактировать");
        buttonEdit.addActionListener(this);
        buttonDelete = new JButton("Удалить");
        buttonDelete.addActionListener(this);

```



```

panelControl.add(buttonShow);
panelControl.add(buttonCreate);
panelControl.add(buttonEdit);
panelControl.add(buttonDelete);
add(panelControl);

//Создание панели "Поиск".
panelFind = new JPanel();
panelFind.setPreferredSize(new Dimension(width_window, 50));
panelFind.setBorder(BorderFactory.createTitledBorder("\Поиск\"));
panelFind.setLayout(new GridLayout());
textFieldFind = new JTextField();
buttonFind = new JButton("Поиск");
buttonFind.addActionListener(this);
panelFind.add(textFieldFind);
panelFind.add(buttonFind);
add(Box.createRigidArea(
    new Dimension(0, 10))); // Отступ сверху вниз на 10 пикселей
add(panelFind);

//Создание панели "Список контактов".
panelShow = new JPanel();
panelShow.setPreferredSize(new Dimension(width_window, 130));
panelShow.setLayout(new BoxLayout(panelShow, BoxLayout.Y_AXIS));
panelShow.setBorder(BorderFactory.createTitledBorder("\Список контактов\"));
add(Box.createRigidArea(
    new Dimension(0, 10))); // Отступ сверху вниз на 10 пикселей
data = new Object[][]{};

tableShowModel = new DefaultTableModel(new Object[]{"ФИО", "Телефон"}, 0){
    // Disabling User Edits in a JTable with DefaultTableModel
    @Override
    public boolean isCellEditable(int row, int column) {
        return false;
    };
};

tableShow = new JTable();
tableShow.setModel(tableShowModel);
panelShow.add(new JScrollPane(tableShow));
labelFindCol = new JLabel("Найдено записей: 0");
panelShow.add(labelFindCol);
add(panelShow);

try {
    conn = DriverManager.getConnection(dbURL, user_name, user_password);
    stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_READ_ONLY);
} catch (SQLException err) {
    System.out.println(err.getMessage());
}

}

private static void createAndShowGUI() {
    JFrame frame = new JFrame("Адресная книга");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    mainFrame = frame;

    JComponent componentPanelAddressBook = new AddressBook();
    frame.setContentPane(componentPanelAddressBook);

    frame.pack();
    frame.setVisible(true);
}

public static void main(String[] args) {
    //Schedule a job for the event-dispatching thread:
    //creating and showing this application's GUI.

```

```

        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        int dataToSize = 7;
        String[] dataTo = new String[dataToSize];

        for (int i = 0; i < dataToSize; i++) {
            dataTo[i] = "";
        }

        if ("Создать".equals(command)) {
            JDialog dialogContact = new JDialog(mainFrame,
                "Новый контакт...", JDialog.DEFAULT_MODALITY_TYPE);

            PanelContact panelContact = new PanelContact(command, dataTo);
            dialogContact.setBounds(
                delta_size_dialog, delta_size_dialog,
                panelContact.getContactPanelWidth() + 3*delta_size_dialog,
                panelContact.getContactPanelHeight() + delta_size_dialog);
            dialogContact.add(panelContact);
            dialogContact.setVisible(true);
        }

        try {
            if (("Редактировать".equals(command) || "Просмотреть".equals(command))
                && rs != null && tableShow.getSelectedRow() > -1) {
                rs.first();
                do {
                    if (rs.getString("PHONENUMBER").
                        equals(tableShowModel
                            .getValueAt(tableShow.getSelectedRow(), 1)
                            .toString())) {
                        dataTo[0] = rs.getString("ID_CONTACTS");
                        dataTo[1] = rs.getString("FIRSTNAME");
                        dataTo[2] = rs.getString("LASTNAME");
                        dataTo[3] = rs.getString("MIDLENAME");
                        dataTo[4] = rs.getString("TELEPHONE");
                        dataTo[5] = rs.getString("ID_PHONENUMBER");
                        dataTo[6] = rs.getString("PHONENUMBER");
                        String title = "";
                        if ("Редактировать".equals(command)) {
                            title = "Изменить контакт...";
                        }
                        if ("Просмотреть".equals(command)) {
                            title = "Просмотреть контакт...";
                        }
                        JDialog dialogContact = new JDialog(mainFrame,
                            title, JDialog.DEFAULT_MODALITY_TYPE);
                        PanelContact panelContact = new PanelContact(command, dataTo);
                        dialogContact.setBounds(
                            delta_size_dialog, delta_size_dialog,
                            panelContact.getContactPanelWidth()
                                + 3*delta_size_dialog,
                            panelContact.getContactPanelHeight()
                                + delta_size_dialog);
                        dialogContact.add(panelContact);
                        dialogContact.setVisible(true);
                        break;
                    }
                } while (rs.next());
            }
        } catch (SQLException err1) {

```

```

        System.out.println(err1.getMessage());
    }

    if ("Поиск".equals(command)){
        findByString(textFieldFind.getText());
    }

    try {
        if ("Удалить".equals(command)
            && rs != null && tableShow.getSelectedRow() > -1) {
            rs.first();
            do {
                if (rs.getString("PHONENUMBER").
                    equals(tableShowModel
                        .getValueAt(tableShow.getSelectedRow(), 1)
                        .toString())) {
                    String SQL_UpdateContacts
                        = "DELETE FROM CONTACTS WHERE TELEPHONE = "
                        + rs.getString("TELEPHONE");
                    String SQL_updatePhoneNumbers
                        = "DELETE FROM PHONENUMBERS WHERE ID_PHONENUMBER = "
                        + rs.getString("ID_PHONENUMBER");
                    PreparedStatement updateContacts = null;
                    PreparedStatement updatePhoneNumbers = null;
                    conn.setAutoCommit(false);
                    updateContacts = conn.prepareStatement(SQL_UpdateContacts);
                    updatePhoneNumbers = conn.prepareStatement(
                        SQL_updatePhoneNumbers);
                    int executeUpdate = updateContacts.executeUpdate();
                    executeUpdate = updatePhoneNumbers.executeUpdate();
                    conn.commit();

                    if (updateContacts != null) updateContacts.close();
                    if (updatePhoneNumbers != null) updatePhoneNumbers.close();
                    conn.setAutoCommit(true);
                    findByString("");
                    break;
                }
            } while (rs.next());
        }
    } catch (SQLException err1) {
        try {
            conn.rollback();
            conn.setAutoCommit(true);
            JOptionPane.showMessageDialog(this,
                "Транзакция на удаление не выполнена.\n"
                + "Смотрите сообщения в консоли.");
            System.out.println(err1.getMessage());
        } catch (SQLException err2) {
            System.out.println(err2.getMessage());
        }
    }
}

private void findByString(String textFind) {
    try {
        while(tableShowModel.getRowCount() > 0) {
            tableShowModel.removeRow(0);
        }
        SQL =
            "SELECT CONTACTS.*, PHONENUMBERS.* "
            + "FROM CONTACTS, PHONENUMBERS "
            + "WHERE CONTACTS.TELEPHONE = PHONENUMBERS.ID_PHONENUMBER "
            + "AND CONTACTS.FIRSTNAME LIKE '"
            + textFind + "%'";
        rs = stmt.executeQuery(SQL);
        while(rs.next()) {
            String fio =
                rs.getString("FIRSTNAME") + " " +

```

```

        rs.getString("LASTNAME") + " " +
        rs.getString("MIDLENAME");
        String phonenum = rs.getString("PHONENUMBER");
        tableShowModel.addRow(new Object[]{fio, phonenum});
    }
    labelFindCol.setText("Найдено записей: " + tableShowModel.getRowCount());
} catch (SQLException err) {
    System.out.println(err.getMessage());
}
}

```

```

class PanelContact extends JPanel implements ActionListener {
    private final int width_window = 300; //Кратно трём
    private final int height_window = 143;
    private final int height_button_panel = 40;
    private final int height_gap = 3;

    private String mode;
    private String dataTo[];

    private JPanel panelUp, panelLabel, panelText, panelButton;

    private JLabel labelFirstName;
    private JLabel labelLastName;
    private JLabel labelMiddleName;
    private JLabel labelPhoneNumber;

    private JTextField txtFieldFirstName;
    private JTextField txtFieldLastName;
    private JTextField txtFieldMiddleName;
    private JTextField txtPhoneNumber;

    private JButton buttonApply;
    private JButton buttonCancel;

    public PanelContact(String mode, String[] dataTo) {
        super();
        this.mode = mode;
        this.dataTo = dataTo;
        setLayout(new BorderLayout(this, BorderLayout.PAGE_AXIS));
        setPreferredSize(new Dimension(width_window, height_window));

        panelUp = new JPanel(); //Панель для размещения панелей
        panelLabel = new JPanel();
        panelText = new JPanel();
        panelButton = new JPanel();

        labelFirstName = new JLabel("Фамилия");
        labelLastName = new JLabel("Имя");
        labelMiddleName = new JLabel("Отчество");
        labelPhoneNumber = new JLabel("Номер тел.");

        txtFieldFirstName = new JTextField(dataTo[1]);
        txtFieldLastName = new JTextField(dataTo[2]);
        txtFieldMiddleName = new JTextField(dataTo[3]);
        txtPhoneNumber = new JTextField(dataTo[6]);

        buttonApply = new JButton("Принять");
        buttonApply.addActionListener(this);
        buttonCancel = new JButton("Отменить");
        buttonCancel.addActionListener(this);

        panelUp.setPreferredSize(new Dimension(width_window, height_window
            - height_button_panel - height_gap));
        panelUp.setBorder(BorderFactory.createBevelBorder(1));
        add(panelUp);
        panelUp.setLayout(new BorderLayout(panelUp, BorderLayout.LINE_AXIS));

        panelLabel.setPreferredSize(new Dimension(width_window/3, height_window

```

```

        - height_button_panel - height_gap));
panelLabel.setBorder(BorderFactory.createBevelBorder(1));
panelLabel.setLayout(new GridLayout(4,1));
panelLabel.add(labelFirstName);
panelLabel.add(labelLastName);
panelLabel.add(labelMiddleName);
panelLabel.add(labelPhoneNumber);

panelText.setPreferredSize(new Dimension(2*width_window/3, height_window
        - height_button_panel - height_gap));
panelText.setBorder(BorderFactory.createBevelBorder(1));
panelText.setLayout(new GridLayout(4,1));
panelText.add(txtFieldFirstName);
panelText.add(txtFieldLastName);
panelText.add(txtFieldMiddleName);
panelText.add(txtPhoneNumber);

panelUp.add(panelLabel);
panelUp.add(panelText);

add(Box.createRigidArea(
    new Dimension(0, height_gap))); // Отступ 10 пикселей

panelButton.setPreferredSize(
    new Dimension(width_window, height_button_panel));
panelButton.setBorder(BorderFactory.createBevelBorder(1));
add(panelButton);
panelButton.setLayout(new FlowLayout());
panelButton.add(buttonApply);
panelButton.add(buttonCancel);

if ("Просмотреть".equals(mode)) {
    buttonApply.setEnabled(false);
    txtFieldFirstName.setEditable(false);
    txtFieldLastName.setEditable(false);
    txtFieldMiddleName.setEditable(false);
    txtPhoneNumber.setEditable(false);
}

}

public int getContactPanelWidth(){
    return width_window;
}

public int getContactPanelHeight(){
    return height_window;
}

@Override
public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();
    JDialog dialog = (JDialog) SwingUtilities.getWindowAncestor(this);
    if ("Отменить".equals(command)) {
        dialog.dispose();
    }
    if ("Принять".equals(command)) {
        if (isTelephone(txtPhoneNumber.getText(), 5, 12)
            && isFromFIO(txtFieldFirstName.getText(), 1, 30)
            && isFromFIO(txtFieldLastName.getText(), 1, 30)
            && isFromFIO(txtFieldMiddleName.getText(), 1, 30)) {
            String SQL_Update_TBL_PhoneNumbers = null;
            String SQL_Update_TBL_Contacts = null;
            if ("Создать".equals(mode)) {
                //Переписать на поиск первого свободного ID
                int r[] = findMaxID();
                SQL_Update_TBL_PhoneNumbers =
                    "INSERT INTO PHONENUMBERS (ID_PHONENUMBER, PHONENUMBER) "
                    + "VALUES (" + (r[0] + 1) + ", "

```

```

        + "'" + txtPhoneNumber.getText() + "'");
SQL_Update_TBL_Contacts = "INSERT INTO CONTACTS"
        + " (ID_CONTACTS, FIRSTNAME,"
        + "  LASTNAME, MIDDLENAME, TELEPHONE) "
        + "VALUES (" + (r[1] + 1) + ", "
        + "'" + txtFieldFirstName.getText() + "', "
        + "'" + txtFieldLastName.getText() + "', "
        + "'" + txtFieldMiddleName.getText() + "', "
        + (r[0] + 1) + ")";
    }
    if ("Редактировать".equals(mode)) {
        SQL_Update_TBL_PhoneNumbers =
            "UPDATE PHONENUMBERS SET "
            + "PHONENUMBER = '" + txtPhoneNumber.getText() + "' "
            + "WHERE ID_PHONENUMBER = " + dataTo[5] + ";";
        SQL_Update_TBL_Contacts =
            "UPDATE CONTACTS SET "
            + "FIRSTNAME = '" + txtFieldFirstName.getText() + "', "
            + "LASTNAME = '" + txtFieldLastName.getText() + "', "
            + "MIDDLENAME = '" + txtFieldMiddleName.getText() + "' "
            + "WHERE ID_CONTACTS = " + dataTo[0] + ";";
    }
    try {
        PreparedStatement createPhoneNumber = null;
        PreparedStatement createContact = null;
        conn.setAutoCommit(false);
        createPhoneNumber = conn.prepareStatement(
            SQL_Update_TBL_PhoneNumbers);
        int executeUpdate = createPhoneNumber.executeUpdate();
        createContact = conn.prepareStatement(SQL_Update_TBL_Contacts);
        executeUpdate = createContact.executeUpdate();
        conn.commit();
        if (createPhoneNumber != null) createPhoneNumber.close();
        if (createContact != null) createContact.close();
        conn.setAutoCommit(true);
    } catch (SQLException err1) {
        try {
            conn.rollback();
            conn.setAutoCommit(true);
            JOptionPane.showMessageDialog(this,
                "Транзакция на создание контакта не выполнена.\n"
                + "Смотрите сообщения в консоли.");
            System.out.println(err1.getMessage());
        } catch (SQLException err2) {
            System.out.println(err2.getMessage());
        }
    }
    findByString("");
    dialog.dispose();
} else {
    JOptionPane.showMessageDialog(this, "Исправьте введённые данные.\n"
        + "Допустимо:\n"
        + "номер телефона - от 5 до 12 цифр,\n"
        + "фамилия, имя, отчество - каждое от 1 до 30 символов.");
}
}

}

private int[] findMaxID() {
    int[] r = {0, 0};
    try {
        SQL =
            "SELECT CONTACTS.*, PHONENUMBERS.* "
            + "FROM CONTACTS, PHONENUMBERS "
            + "WHERE CONTACTS.TELEPHONE = PHONENUMBERS.ID_PHONENUMBER "
            + "AND CONTACTS.FIRSTNAME LIKE '%'";
        rs = stmt.executeQuery(SQL);
        int

```

```

        max_ID_PHONENUMBER = r[0],
        max_ID_CONTACTS = r[1];
    int tmp;
    while(rs.next()) {
        tmp = new Integer(rs.getString("ID_PHONENUMBER"));
        if (tmp > max_ID_PHONENUMBER) max_ID_PHONENUMBER = tmp;
        tmp = new Integer(rs.getString("ID_CONTACTS"));
        if (tmp > max_ID_CONTACTS) max_ID_CONTACTS = tmp;
        r[0] = max_ID_PHONENUMBER;
        r[1] = max_ID_CONTACTS;
    }
} catch (SQLException err) {
    System.out.println(err.getMessage());
}
return r;
}

private Boolean isTelephone(String s, int min_length, int max_length) {
    if (s.length() < min_length || max_length < s.length()) {
        return false;
    }
    for (int i = 0; i < s.length(); i++) {
        if (Character.isDigit(s.charAt(i))) {
        } else {
            return false;
        }
    }
    return true;
}

private Boolean isFromFIO(String s, int min_length, int max_length) {
    if (s.length() < min_length || max_length < s.length()) {
        return false;
    }
    for (int i = 0; i < s.length(); i++) {
        if (Character.isLetter(s.charAt(i))) {
        } else {
            return false;
        }
    }
    return true;
}

}
}
}

```


Практическая часть

1. Запустить учебное приложение. Для этого нужно создать соответствующую ему БД и настроить подключение к ней. Продемонстрировать работу приложения преподавателю.
2. В зависимости от варианта выполняется разработка простого приложения, использующего связку с базой данных.
3. При разработке приложения использовать БД derby.
4. В разрабатываемой БД должно быть не менее двух связанных по ключу таблиц. Далее при описании таблиц эта связка обозначена лишь формально, т. е. вы должны спроектировать вашу БД, желательно показать проект перечня полей в таблицах преподавателю.
5. Клиентский интерфейс доступа к БД:
0 — через GUI клиента (оконное приложение),
1 — через консоль клиента (консольное приложение).
6. Обеспечить сброс к значениям по умолчанию значений всех не ключевых полей на основе данных записанных в:
0 — программе;
1 — файле.
7. Для задания «Учебные аудитории». Вывод всех значений указанного поля (не «ключа») (см. П.8) указанной таблицы в лексикографическом порядке (на консоль, либо в текстовое поле, либо другой удобный для вас графический компонент):
0 — первой,
1 — второй.
8. Поле по порядку таблицы указанной в П.7:
0 — первое,
1 — второе.
9. Возможные базовые (расширение допустимо) варианты данных, хранящихся в БД, приведены в следующих пунктах, также для этих пунктов указаны вариации функций, которые необходимо реализовать.
10. Учебные аудитории (табл.1: {«уникальный ключ», учебное здание, номер аудит., наименование, площадь}, табл.2: {«уникальный ключ», ФИО ответственного, должность, телефон, возраст});
(а : 000) найти общую площадь в указанном здании, вывести список аудиторий в указанном здании;
(б : 001) найти средний возраст ответственных, найти общую площадь за указанным ответственным;
(в : 010) вывести список аудиторий в указанном здании, найти общую площадь аудиторий закреплённых за указанным ответственным;
(г : 011) вывести список аудиторий в указанном здании, вывести список аудиторий закреплённых за указанным ответственным;
(д : 100) вывести телефонный справочник (ФИО, телефон) в лексикографическом порядке, найти среднюю площадь закреплённую за ответственными;
(е : 101) вывести штатное расписание (должность, ФИО) в лексикографическом порядке, найти общую площадь за указанным ответственным;
(ё : 110) вывести список самых младших сотрудников (ФИО, должность, телефон, возраст) в лексикографическом порядке, найти общую площадь аудиторий закреплённых за указанным ответственным;
(ж : 111) вывести список самых старших сотрудников (ФИО, должность, телефон, возраст) в лексикографическом порядке, вывести список сотрудников в порядке возрастания суммы площадей, которые за ними закреплены;
11. Для задания «Домашняя библиотека». Вывод всех значений указанного поля (не «ключа») (см. П.12) указанной таблицы в лексикографическом порядке (на консоль,

либо в текстовое поле, либо другой удобный для вас графический компонент):

0 — первой,

1 — второй.

12. Поле по порядку таблицы указанной в П.11:

0 — второе,

1 — третье.

13. Домашняя библиотека (табл.1: {«уникальный ключ», этаж, шкаф, полка}, табл.2:

{«уникальный ключ», Автор (ФИО), название издания, издательство, год издания, количество страниц, год написания, вес в граммах});

(а : 000) вывести авторов (ФИО) на указанной полке в лексикографическом порядке, вывести суммарное количество изданий (Автор (ФИО), название издания, издательство, год издания, количество страниц) указанного автора;

(б : 001) вывести издания в указанном шкафу в лексикографическом порядке, вывести суммарное количество страниц в изданиях указанного автора;

(в : 010) вывести авторов в указанном шкафу в лексикографическом порядке, вывести суммарный вес изданий в указанном шкафу;

(г : 011) вывести издания на указанном этаже в лексикографическом порядке, вывести издания, для которых год издания существенно (более 10 лет) отличается от даты написания;

(д : 100) вывести авторов на указанном этаже в лексикографическом порядке, вывести издания, для которых, не указан год написания;

(е : 101) вывести издательства на указанной полке в лексикографическом порядке, вывести самое тяжёлое и самое лёгкое по весу издание в указанном шкафу;

(ё : 110) вывести издательства в указанном шкафу в лексикографическом порядке, вывести самое короткое и самое длинное по количеству страниц издание на указанном этаже;

(ж : 111) вывести издательства на указанном этаже в лексикографическом порядке, вывести издания указанного автора в хронологическом порядке их написания.

14. *Ваше фактическое задание выполнять применительно к БД:*

0 — «Учебные аудитории»,

1 — «Домашняя библиотека».

15. Выбор записи для удаления или изменения по:

0 — порядковому номеру в общем списке в приложении,⁶

1 — по уникальному ключу (смотреть в БД).⁷

16. В приложении реализовать просмотр записей, хранящихся в БД, добавление записей в БД, изменение записей, хранящихся в БД, и удаление записей из БД.

17. Продемонстрировать использование транзакций (несколько команд, которые либо применяются все либо происходит откат выполненных).

6 Если приложение консольное, то номер записи (и связанных с ней записей) может идти отдельной колонкой; если приложение графическое, то допускается выбор записи либо мышью (как в учебном приложении) либо другим удобным для вас интерактивным способом.

7 Если приложение графическое, то добавить на графический интерфейс приложения интерактивный элемент для выбора (или ввода) уникального ключа.

Варианты заданий

Таблица 9.2. Задания на лабораторную работу

№	П.5	П.6	П.7	П.8	П.10, П.13	П.12	П.14	П.15
1	1	1	1	1	101	1	1	0

Число в указанном варианте следует толковать как двоичное число каждый разряд которого соответствует столбцу из таблицы 1.

Например, первый вариант определяется представлением числа 1006 в двоичной системе счисления.

1	1006	27	750	53	184	79	394
2	606	28	258	54	392	80	724
3	626	29	206	55	94	81	964
4	967	30	366	56	928	82	436
5	109	31	162	57	304	83	324
6	70	32	900	58	310	84	168
7	371	33	337	59	143	85	660
8	863	34	956	60	229	86	216
9	122	35	667	61	970	87	289
10	994	36	746	62	9	88	424
11	924	37	575	63	161	89	951
12	288	38	840	64	646	90	688
13	965	39	243	65	263	91	518
14	321	40	817	66	318	92	408
15	396	41	336	67	59	93	828
16	530	42	967	68	934	94	793
17	40	43	632	69	523	95	294
18	261	44	255	70	983	96	733
19	482	45	588	71	929	97	788
20	683	46	417	72	953	98	455
21	436	47	139	73	809	99	662
22	34	48	112	74	910	100	867
23	602	49	984	75	966	101	672
24	131	50	396	76	166		
25	544	51	835	77	402		
26	753	52	919	78	58		